170, which is usually designed for general purposes, does not take into account the internal structure and bottleneck of various components in memory system 104.

Normally, memory manager 165 performs its tasks in the background, e.g., independent from and/or in parallel with system processor 105 and operating system 170. As memory manager 165 does not use processor 105, processor bus 1005, or other processor resources, memory manager 165 does not interfere with processor 105's performance. In one embodiment, memory manager 165, through memory table 165, uses physical address on system bus 1010 to locate the data. In many situations, this physical address was translated from a virtual address.

THE MEMORY TABLE

Memory table 160 includes entries to locate the requested data stored in different storage areas. In one embodiment, memory table 160 is part of memory controller 110. However, memory table 160 can be at any convenient locations such as in a memory unit, physical memory, main memory, cache, part of the processor, etc. Further, memory manager 165 manages memory table 160. Processor 105, translation look-aside buffer 150, or operating system 170 do not need to know that memory table 160 exists. This is because, in one embodiment, memory system 104, receiving the physical address on system bus 1010, returns the accessed data with the same physical address. In one embodiment, memory table 160 is implemented in hardware such as in random access memory (RAM) or memory controller 110, which normally can be run at high speed and thus does not add significant delay to a memory access.

FIG. 2 shows a memory table 200 as one embodiment of memory table 160. Table 200, being in use with memory system 104, includes a plurality of table entries, e.g., 210-1 to 210-N for N entries. If there is no reference to an actual data block, an entry

210 is "NIL." However, if a data block has been allocated, a corresponding entry 210 points to that data block. In this FIG. 2 example, entries 210-2, 210-3, 210-4, 210-5 point to data blocks located at various random locations in both physical memory 120 and hard disc 130. Further, physical memory 120 and hard disc 130 are shown as only an example, the data blocks can be in various different storage areas, in accordance with the techniques disclosed herein.

ENTRIES OF MEMORY TABLE

Normally, memory table 160 includes enough entries 210 to cover all data blocks of memory system 104 as seen by processor 105 and operating system 170. For example, memory table 160 contains 1.64 million entries covering 1.64 million blocks resulted from a 1.64G memory system 104 having each data block of 1K. Further, each entry 210 corresponds to a block covering a physical address range. For example, entry 1 corresponds to block 1 covering physical addresses 0 to 1023, entry 2 corresponds to block 2 covering physical addresses 1024 to 2047, and entry 3 corresponds to block 3 covering physical addresses 2048 to 3071, etc. Additionally, 22 bits, e.g., bits 10 to 31 of physical address on system bus 1010 are used to perform a translation lookup, i.e., to find an entry, in memory table 160. The example uses a simple addressing scheme to perform the translation table look-up, but, in accordance with the technique disclosed herein, a hash table or any other effective method can be used for such a translation lookup. Bits 10 to 31 actually address 2 million blocks; processor 105 should not send a memory access that is beyond the range of translation table 160. If processor 105 does send such an access, then translation table 160 marks the access as invalid.

In one embodiment, each entry 210 includes a "valid" bit, an "updating location bit," and a "static" bit. The valid bit indicates whether the data block pointed to by the

corresponding entry has been initialized. In one embodiment, if the data block is initialized, then the valid bit is set, e.g., having a logic "one" value, and if the data block is not initialized, then the valid bit is reset, e.g., having a logic "zero" value. The updating-location bit indicates whether an access to the data block is allowable. For example, if a table entry 210 is being modified or if the data block is being in transit from one location to another location, then the updating-location bit for that entry 210 is set, and a memory access to the data block is buffered during the time this updating-location bit is set. When this updating-location bit is reset, e.g., the table entry 210 is completely modified and the block is settled in its location, accessing the block is then allowable, and any buffered accesses are completed. The static bit indicates whether a particular data block must always be in physical memory 120, and cannot moved to any other location. In one embodiment, if the static bit is set, then the data block cannot be moved out of physical memory 120. In one embodiment, the valid bit can be replaced by using an invalid entry located in the memory table entry. Memory manager 165 detects invalid entries by testing the address stored in the memory table entry instead of using a special bit.

In one embodiment, entries 210 also store statistical information about their corresponding data blocks, such as how long the data blocks have been staying at a particular location, the number of time the block has been accessed during some time period, etc. Memory manager 165 then uses the statistical information accordingly, e.g., to determine when and where to move the data blocks, etc.

TRANSLATING FROM A VIRTUAL ADDRESS TO A LOCATION ADDRESS

Referring to FIG. 3 for an illustration of converting a virtual address on processor bus 1005 in FIG. 1 to a physical address on system bus 1010, and then to a location

address on memory bus 1015, upon a memory access. A location address identifies a data block. In this example, the virtual address is 48 bits, memory system 104 is assigned 32 bits representing 1.64Gb from addresses 0 to 1.64Gb - 1 seen by processor 105 and operating system 170. A page is 16K while a data block is 1K.

Box 304 shows the 48-bit virtual address represented by bits 0-47 in which bits 0-13 represent a 16K page. Each representation of bits 0-13 serves as an offset for accessing the page. Thirty four bits 14-47 represent the virtual page numbers covering all 2**34 pages in the virtual address space.

Box 308 shows a 40-bit physical address, represented by bits 0-39. In other approaches, these bits are used to directly access data in physical memory 120. Each representation of bits 0-13 also serves as an offset for accessing data in a 16K page. Bits 14-39 are translated from bits 14-47 in box 304, using translation look-aside buffer 150.

Box 312 shows a 32-bit location address used for finding data blocks covered by 1.64Gb memory system 104. Ten bits, e.g., bits 0-9, are used to locate data in a data block of 1K. Bits 10-31, being converted from bit 10-31 in box 308, are used to lookup in memory table 160 to identify the location of a data block. In the example of memory system 104, if bits 10-31 are translated to a location address from 0 to 128M-1, 128M to 640M - 1, and 640M to 1.64GB - 1, then the data block is in physical memory 120, hard disc 130, and other storage areas, respectively. Bits 0-9 are then used to convert to a location address in the identified data block.

CREATION OF THE MEMORY TABLE AND DATA BLOCKS

In one embodiment, memory table 160 implemented in hardware is created when the computer system is designed, and memory manger 165 initializes it at boot time. In this embodiment, a certain amount of hardware space is allocated for table 160 and for